SOCKET 编程实验之
简易聊天系统

课程名称：计算机网络

学院：计算机科学技术学院

专业：计算机科学与技术

姓名：孙瑜

学号：17307130312

一、需求分析

1、项目简介

本次课程设计利用 SocketAPI 实现了简易的聊天系统功能，并实现了服务端与客户端的连接与通信，支持多线程。实现语言为 python，可视化界面使用 python 自带的 tkinter 包。

2、功能分析

- 登录功能：输入用户名和正确的密码可以进入聊天系统。
- 注册功能：自定义用户名和密码可以进行注册，注册成功后重新登录进入聊天系统。
- 查看在线用户：在界面左边显示所有在线用户。
- 群聊功能：一个用户发送消息，所有在线用户都可以收到。
- 私聊功能：只有聊天对象才能收到，其余在线用户无法收到。
- 聊天时间功能：收到的每一条消息都附加了消息发送时间。
- 发送表情：可以从表情库中选择表情发送
- 发送图片：从电脑中选择图片发送，客户端和服务端会产生图片缓存
- 发送文件：从电脑中选择文件发送，客户端自动缓存文件，其他用户可以点击立即打开文件
- 菜单功能：可以点击查看缓存在本地文件夹中的图片和文件。
- 离线功能：点击菜单中的退出或者关闭界面即默认退出聊天室。

二、附件清单

- client.py：客户端及可视化界面的实现
- server.py：服务端的实现
- client_file_cache 文件夹：客户端文件缓存，内容会随着程序运行有所改变。
- server_file_cache 文件夹：服务端文件缓存，内容会随着程序运行有所改变。
- client_photo_cache 文件夹：客户端图片缓存，内容会随着程序运行有所改变。
- server_photo_cache 文件夹：服务端图片缓存，内容会随着程序运行有所改变。
- sample 文件夹：发送图片和文件用到的样例
- emoji 文件夹：7 张表情图片
- icon 文件夹：聊天界面按钮用到的图片
- icon.png：聊天器的左上角图标

三、使用说明

打开终端到 Data_Crawler 文件夹下，运行：python server.py, python client.py（client 可以多次运行，server 只运行一次）。

其中已存在的用户名及密码为：{Julia: sunyu}, {Meggie: maiji}。可以用上面两个账户登录，也可自己注册。具体功能见“五、结果展示”。

四、实现方法

本次课程设计中，我完成了 client.py 和 server.py 两个 Python 文件，分别代表客户端和服务端。其中客户端可以重复运行，但是服务端只能运行一次。这一部分将分别对两个文件中各个功能的代码进行展示和解释，对于运行出来的可视化界面呈现在“三、运行结果”中。

1、client.py

本文件既实现了与服务端连接通信，也利用了 tkinter 实现了可视化界面。可视化界面使用了阿里巴巴矢量图库部分 icon（https://www.iconfont.cn/）。

1）登录功能

① 可视化界面

```
# 登录界面
root1 = tkinter.Tk()
root1.title('用户登录')
p = Image.open('icon.png')
icon = ImageTk.PhotoImage(p)
root1.tk.call('wm', 'iconphoto', root1._w, icon)
root1['height'] = 170
root1['width'] = 300

root1.resizable(0, 0)  # 限制窗口大小
```

```
# 用户名
label1 = tkinter.Label(root1, text='用户名:')
label1.place(x=20, y=30, width=100, height=20)
entry1 = tkinter.Entry(root1, width=80)
entry1.place(x=120, y=30, width=130, height=20)
# 密码
label2 = tkinter.Label(root1, text='密码:')
label2.place(x=30, y=60, width=80, height=20)
entry2 = tkinter.Entry(root1, width=80)
entry2.place(x=120, y=60, width=130, height=20)
entry2['show'] = '*'
```

```
root1.bind('<Return>', login)  # 回车绑定登录功能
loginbut = tkinter.Button(root1, text='登录', command=login)
loginbut.place(x=70, y=110, width=60, height=25)
signinbut = tkinter.Button(root1, text='注册', command=signin)
signinbut.place(x=160, y=110, width=60, height=25)

root1.mainloop()
```

首先写了登录界面，自定义了 icon。由于窗口中的插件是固定位置的，所以我设置了不可调节窗口大小。然后设置了用户名和密码的输入框，并将其中的输入内容保存到变量中，以便于后面判断用户名和密码是否匹配，密码设置了不可视。最后增加了回车绑定登录功能。

② 登录函数

```
user_name = ''  # 当前用户
users = {'Julia': ['sunyu', '127.0.0.1:50001'], 'Meggie': ['maiji', '127.0.0.1:50001']}  # 用户列表
```

首先定义了一个 user_name 指示当前变量。然后定义了一个 users 字典，其中 key 为用户名，value 是一个[密码，IP:PORT]的列表，用于将每个用户名的对应信息关联起来。

```
# 登录按钮
def login(*args):
    global IP, PORT, user_name
    user_name = entry1.get()
    password = entry2.get()
    if user_name in users:
        if users[user_name][0] == password:
            IP, PORT = users[user_name][1].split(':')
            PORT = int(PORT)
            root1.destroy()
        else:
            tkinter.messagebox.showerror('密码错误', message='密码错误！\n请检查你的密码输入')
    else:
        tkinter.messagebox.showerror('用户名错误', message='用户名不存在！\n请检查你的用户名输入，或者点击注册进行用户创建')
```

首先从界面 get 到输入的用户名和密码。先判断 user_name 是否是 users 的一个 key，如果不是，则弹出警告说明用户名不存在；如果在，则判断密码与用户名是否匹配，如果不匹配，则弹出警告说明密码错误。如果成功匹配了用户名和密码，则将当前用户的 IP 和 PORT 读取到变量中，然后关闭登录界面。

③ 与 server 建立连接之传输用户名

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((IP, PORT))
if user_name:
    s.sendall(user_name.encode('utf-8'))  # 发送用户名
else:
    s.sendall('no'.encode('utf-8'))  # 没有输入用户名则标记no

# 如果没有用户名则将ip和端口号设置为用户名
addr = s.getsockname()  # 获取客户端ip和端口号
addr = addr[0] + ':' + str(addr[1])
if user_name == '':
    user_name = addr
```

第一次调用 socket 包，建立 socket 对象，定义为 TCP 类型，根据当前用户的 IP 和 PORT 建立连接，然后调用 sendall 将用户名发给 server。这里允许用户注册的时候不填用户名，这样内部会自动将当前用户名定义为 IP+PORT，以便于后面消息显示发送方。

2）注册功能

在登录界面还定义了一个注册按钮，点击后会进入 signin 函数，以下是 signin 函数中的内容：

① 界面

```
global users
# 注册界面
root2 = tkinter.Tk()
root2.title('用户注册')
root2['height'] = 200
root2['width'] = 300
root2.resizable(0, 0)  # 限制窗口大小

# 用户名
label_1 = tkinter.Label(root2, text='用户名')
label_1.place(x=20, y=30, width=100, height=20)
entry_1 = tkinter.Entry(root2, width=80)
entry_1.place(x=120, y=30, width=130, height=20)
```

```
# 密码
label_2 = tkinter.Label(root2, text='密码')
label_2.place(x=30, y=60, width=80, height=20)
entry_2 = tkinter.Entry(root2, width=80)
entry_2.place(x=120, y=60, width=130, height=20)
entry_2['show'] = '*'
# 确认密码
label_3 = tkinter.Label(root2, text='密码确认')
label_3.place(x=30, y=90, width=80, height=20)
entry_3 = tkinter.Entry(root2, width=80)
entry_3.place(x=120, y=90, width=130, height=20)
entry_3['show'] = '*'

root2.bind('<Return>', forsignin)  # 回车绑定登录功能
but = tkinter.Button(root2, text='注册', command=forsignin)
but.place(x=110, y=140, width=60, height=25)
```

类似登录功能，注册也先写界面，只是多了一个确认密码的输入框，做到与真实情况相近。

② 注册函数

```
def forsignin(*args):
    new_user = entry_1.get()
    new_pass = entry_2.get()
    new_pass_con = entry_3.get()
    if new_user in users:
        tkinter.messagebox.showerror('注册失败', message='用户名已存在！\n请重新输入用户名或直接返回页面登录')
    elif new_pass != new_pass_con:
        tkinter.messagebox.showerror('注册失败', message='两次密码输入不一致！\n请检查你的密码输入')
    else:
        users[new_user] = [new_pass, '127.0.0.1:50001']
        tkinter.messagebox.showinfo('注册成功', message='注册成功！\n请在登录界面重新登录')
        root2.destroy()
```

同样在获取界面输入的三个内容后进行判断。如果新注册的用户名已经存在则会弹出警告说明用户名已存在；如果密码与确认密码两变量内容不一致，则会弹出警告说明说明密码输入不一致。最后注册成功会关闭注册界面。

3）聊天界面

聊天界面是聊天器的主要部分，本部分将分别从发送信息和和接受信息解释，而每部分再根据信息类型（文本、表情、图片、文件）进行分类。

① 界面实现

```
# 聊天界面
root = tkinter.Tk()
root.title(user_name)  # 窗口命名为用户名
root['height'] = 500
root['width'] = 680
root.resizable(0, 0)  # 限制窗口大小
p = Image.open('icon.png')
icon = ImageTk.PhotoImage(p)
root.tk.call('wm', 'iconphoto', root._w, icon)
```

此部分为定义一个新的窗口，大小固定，定义名字为当前用户名

② 发送信息部分

➢ 文本发送框

```
# 创建文本输入框
text_v = tkinter.StringVar()
text_v.set('')
textbox = tkinter.Text(root, font=('微软雅黑', 10))
textbox.place(x=190, y=330, width=480, height=155)
```

```
# 创建发送按钮
text = tkinter.Button(root, text='发送', command=send, bd=0)
text.place(x=590, y=450, width=50, height=25)
root.bind('<Return>', send)
```

首先用用 tkinter.Text 建立一个文本输入框，放在合适的位置，然后创建发送键，让点击发送键和回车都关联到 send 函数。函数定义如下：

```
# 发送
def send(*args):
    onlinde_users.append('-------Group chat--------')
    print(chat)
    if chat not in onlinde_users:
        tkinter.messagebox.showerror('发送错误', message=chat + '不在线或不存在')
    if chat == user_name:
        tkinter.messagebox.showerror('发送错误', message='不能发送消息给自己！')
        return
    date = time.strftime('%Y-%m-%d %H:%M:%S')
    mes = ' (' + date + ') ' + textbox.get('0.0', 'end') + ':;' + user_name + ':;' + chat  # 添加聊天对象标记
    # print(mes)
    s.sendall(mes.encode('utf-8'))
    textbox.delete('0.0', 'end')  # 发送后清空文本框
```

首先将 Group Chat 添加到在线用户的列表中，表明只要上线就默认进行群聊。这里的 chat 是指聊天对象，由于之后还有私聊功能，所以添加了一个判断聊天对象是否在线的判断。同时规定不能给自己发送消息。

如果聊天对象合理，则记录下当前的时间为消息发送时间。然后定义发送给 server 的消息组成为：时间＋文本输入框内容+:;＋发送消息的用户＋:;＋接收消息的对象。之后将封装好的文本消息传送给 server 端，同时清除文本输入框所有内容，以便于下次输入。

➢ 私聊功能：

```
# 私聊功能
def private(*args):
    global chat
    online_indexs = userlist.curselection()  # 选择进行私聊的对象的索引
    online_index = online_indexs[0]
    if online_index > 0:
        chat = userlist.get(online_index)
        if chat == '-------Group chat--------':
            root.title(user_name)
            return
        pri_title = user_name + '->' + chat
        root.title(pri_title)


userlist.bind('<ButtonRelease-1>', private)
```

当前用户在在线用户列表（解释见③接受信息部分）中选择想要私聊的对象，然后 private 函数会在 online_users 里定位到选择的用户，然后修改聊天对象 chat，同时修改窗口标题为“当前用户->聊天对象”。

- 发送表情按钮：

```
# 定义7个表情，使用全局变量，方便创建和销毁
e1 = ''
e2 = ''
e3 = ''
e4 = ''
e5 = ''
e6 = ''
e7 = ''
# 将图片打开存入变量中
p1 = tkinter.PhotoImage(file='./emoji/愉快.png')
p2 = tkinter.PhotoImage(file='./emoji/可爱.png')
p3 = tkinter.PhotoImage(file='./emoji/爱你.png')
p4 = tkinter.PhotoImage(file='./emoji/笑哭.png')
p5 = tkinter.PhotoImage(file='./emoji/大哭.png')
p6 = tkinter.PhotoImage(file='./emoji/流汗.png')
p7 = tkinter.PhotoImage(file='./emoji/难受.png')
# 用字典将标记与表情图片一一对应，用于后面接收标记判断表情贴图
dic = {'aa**': p1, 'bb**': p2, 'cc**': p3, 'dd**': p4, 'ee**': p5, 'ff**': p6, 'gg**': p7}
eflag = 0  # 判断表情面板开关的标志
```

首先做好预处理工作，定义 7 个表情按钮的全局变量，方便在点击后进行销毁。另外将 7 个表情的图片打开存入变量中，并利用字典变量将名称与图片匹配，以便于后面根据接收到的名字进行显示。

```
# 创建表情按钮
e_p = tkinter.PhotoImage(file='emoji.png')
emoji = tkinter.Button(root, image=e_p, command=rollout, bd=0)
emoji.place(x=200, y=295, width=30, height=30)
```

上图为定表情按钮，对应 rollout 函数如下：

```
def rollout():
    global e1, e2, e3, e4, e5, e6, e7, eflag
    if eflag == 0:
        eflag = 1
        e1 = tkinter.Button(root, command=e_1, image=p1, relief=tkinter.FLAT, bd=0)
        e1.place(x=148, y=225)
        e2 = tkinter.Button(root, command=e_2, image=p2, relief=tkinter.FLAT, bd=0)
        e2.place(x=183, y=225)
        e3 = tkinter.Button(root, command=e_3, image=p3, relief=tkinter.FLAT, bd=0)
        e3.place(x=218, y=225)
        e4 = tkinter.Button(root, command=e_4, image=p4, relief=tkinter.FLAT, bd=0)
        e4.place(x=253, y=225)
        e5 = tkinter.Button(root, command=e_5, image=p5, relief=tkinter.FLAT, bd=0)
        e5.place(x=148, y=260)
        e6 = tkinter.Button(root, command=e_6, image=p6, relief=tkinter.FLAT, bd=0)
        e6.place(x=183, y=260)
        e7 = tkinter.Button(root, command=e_7, image=p7, relief=tkinter.FLAT, bd=0)
        e7.place(x=218, y=260)
    else:
        e1.destroy()
        e2.destroy()
        e3.destroy()
        e4.destroy()
        e5.destroy()
        e6.destroy()
        e7.destroy()
        eflag = 0
```

rollout 中，先判断 eflag 是否等于零，即表情面板是否已经展开，如果未展开则定义 7 个表情，否则将展开的表情 destroy 掉。

```
def mark(exp):  # 参数是发的表情图标记，发送后将按钮销毁
    global eflag
    mes = exp + ':;' + user_name + ':;' + chat
    s.sendall(mes.encode('utf-8'))
    e1.destroy()
    e2.destroy()
    e3.destroy()
    e4.destroy()
    e5.destroy()
    e6.destroy()
    e7.destroy()
    eflag = 0
```

点击任意表情后，实现的函数为 mark。即将表情信息进行封装：表情代号＋:;＋用户名＋:;＋发送对象，然后发送给 server 端，收起表情面板。

③ 接受信息部分

```
if (t_text in dic) or if_photo[0] == '``' or if_photo[0] == '++':
    # 用不同个颜色区分私聊or群聊or自己发的消息
    message4 = '\n' + message2 + ': '  # \n发送消息的用户:
    date = time.strftime('%Y-%m-%d %H:%M:%S')
    message4 = message4 + ' (' + date + ') '
    if 'Group chat' in message3:
        if message2 == user_name:
            mesbox.insert(tkinter.END, message4, 'blue')
        else:
            mesbox.insert(tkinter.END, message4, 'green')
    elif message2 == user_name or message3 == user_name:
        if message2 == user_name:
            mesbox.insert(tkinter.END, message4, 'grey')
        else:
            mesbox.insert(tkinter.END, message4, 'red')
```

```
else:
    if 'Group chat' in message3 or message2 == user_name or message3 == user_name:
        mesbox.image_create(tkinter.END, image=dic[t_text])
```

在接收信息部分，首先将收到消息的时间打印出来，并判断当前用户是否是聊天对象，从而判断是否需要或者以什么颜色在消息接收框中打印出消息发送者及信息。接着判断在进行了是否为图片和文件的判断后，如果都不是，说明是表情，则在当前用户符合接收条件的情况下将表情显示出来。

4）发送与接收图片

➢ 发送图片功能

```
def picture():
    fileName = tkinter.filedialog.askopenfilename(title='请选择一张图片')
    if fileName:
        sendPhoto(fileName)


# 创建图片按钮
i_p = tkinter.PhotoImage(file='photo.png')
photo = tkinter.Button(root, image=i_p, command=picture, bd=0)
photo.place(x=235, y=295, width=30, height=30)
```

定义一个发送图片的按钮，点击后会弹出选择文件的文件对话框，选中后，将文件名（含路径）传递给 sendPhoto 函数如下：

```
# 将图片上传到图片服务端的缓存文件夹中
def sendPhoto(fileName):
    PORT1 = 50002  # 图片服务器端口号
    s1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s1.connect((IP, PORT1))
    print(s1, ': ', fileName)
    picture = fileName.split('/')[-1]
    print(picture)
    message = 'put ' + picture
    s1.sendall(message.encode('utf-8'))
    time.sleep(0.1)
    print('正在上传图片')
```

```
with open(fileName, 'rb') as f:
    while True:
        a = f.read(1024)
        print(a)
        if not a:
            break
        s1.sendall(a)
    time.sleep(3)  # 延时确保文件发送完整
    s1.sendall('EOF'.encode('utf-8'))
    print('图片上传成功')
s1.sendall('quit'.encode('utf-8'))
time.sleep(1)
# 上传成功后发信息给所有的客户端
mes = '``#' + picture + ':;' + user_name + ':;' + chat
s.sendall(mes.encode('utf-8'))
```

建立一个传输图片的端口号，IP 保持不变，重新建立一个新的 socket 变量，专门用于图片传输。接着从 fileName 中提取图片名称，然后封装第一个图片消息：put＋ ＋图片名，传递给 server 表示 client 即将向 server 上传图片信息；接着封装第二个图片消息：图片的二进制串，server 读取到二进制串后可以重新写成一个新的图片放入缓存，以便于发送给 client，信息要以 EOF 结尾，表示图片上传结束；再次，封装第三个图片信息：quit，表示结束图片上传；最后封装一条描述信息：``#＋图片名＋:;＋用户名＋:;＋聊天对象。

这里注意在上传图片二进制信息的时候要制造延时，否则大图片会有上传不完全的问题。

- 接受图片功能

```
if if_photo[0] == '``':
    mesbox.insert(tkinter.END, '\n')
    recievePhoto(if_photo[1])
    send_photo = './client_photo_cache/' + if_photo[1]
    if 'Group chat' in message3 or message2 == user_name or message3 == user_name:
        # img = tkinter.PhotoImage(file=send_photo)
        img = Image.open(send_photo)
        img = ImageTk.PhotoImage(img)
        mesbox.image_create(tkinter.END, image=img)
```

当判断#前面为``时，说明该信息为图片信息，需要调用 receivePhoto 函数来将服务端的图片缓存下载到本地客户端图片缓存文件夹中，之后如果自己是接收对象，则在消息框中插入缓存下来的图片。

```
# 接收图片
def recievePhoto(fileName):
    PORT1 = 50002
    s1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s1.connect((IP, PORT1))
    message = 'get ' + fileName
    s1.sendall(message.encode('utf-8'))
    fileName = './client_photo_cache/' + fileName
    print('正在下载图片')
    with open(fileName, 'wb') as f:
        while True:
            pic = s1.recv(1024)
            if pic == 'EOF'.encode('utf-8'):
                print('下载成功！')
                break
            f.write(pic)
    time.sleep(1)
    s1.sendall('quit'.encode('utf-8'))
```

在 receivePhoto 文件中，首先定义一个专门用于传输图片的端口号 PORT1，重新建立传输图片的 socket 变量。然后封装 message 为 get++文件名，发送给服务端，指示该客户端要下载指定文件。然后用 open 以二进制写的模式打开新建在 client_photo_cache 缓存文件夹下的图片文件，从服务端接收文件二进制信息写入该图片，最后以接到 EOF 为停止标志。最后下载缓存图片成功，返回给服务端一个 quit 命令表示 get 结束。

5）上传与下载文件

文件的上传与下载功能大致与图片类似，即创建按钮、选择文件、发送 put 命令给 server 指示上传指定文件到服务器缓存、发送 get 指示下载指定文件到客户端缓存。以上功能在此不详细阐述，代码详细见 python 文件。

唯一不同的是，由于文件无法直接在消息框中显示，所以以消息形式打印在消息框中，说明某用户发送了文件。然后弹出一个选择框，提醒收到文件，询问是否打开。如果是，则直接从本地文件缓存中打开该文件，否则可以通过菜单按钮中的“查看文件”来查看所有收到的文件。

```
elif if_photo[0] == '++':
    mesbox.insert(tkinter.END, '\n')
    recieveFile(if_photo[1])
    send_file = './client_file_cache/' + if_photo[1]
    if 'Group chat' in message3 or message2 == user_name or message3 == user_name:
        file_icon = './download.png'
        icon_f = Image.open(file_icon)
        img = ImageTk.PhotoImage(icon_f)
        mesbox.image_create(tkinter.END, image=img)
        mes = '发送文件：' + if_photo[1]
        mesbox.insert(tkinter.END, mes)
        file_box = tkinter.messagebox.askyesno(title='收到文件', message=message2 +
        if file_box:
            # 从缓存区打开该文件
            os.system("start %s" % send_file)
```

2、server.py

1）主函数

```
if __name__ == '__main__':
    chat_server = ChatServer(PORT)
    chat_server.start()
    photo_server = PhotoServer(PORT + 1)
    photo_server.start()
    file_server = FileServer(PORT + 2)
    file_server.start()
```

```
while True:
    time.sleep(1)
    if not chat_server.isAlive():
        print("Chat Server线程不存在")
        sys.exit(0)
    if not photo_server.isAlive():
        print("Photo Server线程不存在")
        sys.exit(0)
    if not file_server.isAlive():
        print("File Server线程不存在")
        sys.exit()
```

在主函数中，分别定义三种消息的线程类，并启动，如果有线程不存在，则退出。注意这里三种消息的 PORT 不同

2）三个 Server 线程类

- ChatServer 消息类

```
class ChatServer(threading.Thread):
    """重写聊天消息线程类，继承threading.Thread"""
    global online_users, mes_q, lock

    def __init__(self, port):
        threading.Thread.__init__(self)
        self.ADDR = ('127.0.0.1', port)
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
def run(self):  # 启动线程
    self.s.bind(self.ADDR)
    self.s.listen(5)
    print('正在运行Chat Server的线程')
    t = threading.Thread(target=self.sendMessage)
    t.start()
    while True:
        conn, addr = self.s.accept()
        print('conn=', conn, 'addr=', addr)
        t1 = threading.Thread(target=self.recieveMessage, args=(conn, addr))
        t1.start()
    self.s.close()
```

首先在 init 里面重新初始化相关参数，定义消息类的 IP+PORT 地址，并定义根据地址定义 socket 变量。接着在启动线程的函数中，将 socket 变量与地址关联，同时最多允许接收 5 个。调用 sendMessage 函数来将服务端的消息传送给客户端，再调用 receiveMessage 函数来接收来自客户端的消息，并且只在有消息传来的时候才会调用改函数。

① sendMessage 函数

```python
def sendMessage(self):
    while True:
        if not mes_q.empty():
            mes_send = ''
            message = mes_q.get()  # 取出消息队列的第一个元素
            if isinstance(message[1], str):  # 判断message[1]是否为str类型
                for i in range(len(online_users)):
                    for j in range(len(online_users)):
                        # 在所有在线用户列表中寻找当前消息的接受者
                        if message[0] == online_users[j][2]:
                            print('消息来自用户[{}]'.format(j))
                            mes_send = ' ' + online_users[j][1] + ': ' + message[1]
                            break
                    print(online_users[i][0])
                    print(mes_send)
                    online_users[i][0].send(mes_send.encode('utf-8'))
            if isinstance(message[1], list):
                print(message[1])
                mes_send = json.dumps(message[1])
                # print('list:', mes_send)
                for i in range(len(online_users)):
                    try:
                        online_users[i][0].send(mes_send.encode('utf-8'))
                    except:
                        pass
```

首先判断消息队列是否为空，即是否有消息需要传送给客户端。如果有，则取出消息队列的第一条消息，并判断消息类型。如果是 str 类型，说明是一条文本消息；如果是 list 类型，说明是在线用户列表。

对于文本消息的情况，对每一个在线用户，在所有用户列表中寻找该文本消息的接收者，然后封装文本消息成消息接收者+消息，然后通过 socket 发送给客户端。

对于在线用户列表的情况，用 json 进行封装，并传递消息给所有在线用户。

② receiveMessage 函数

```python
def recieveMessage(self, conn, addr):
    send_user = ''
    send_user = conn.recv(1024)
    send_user = send_user.decode('utf-8')
    for i in range(len(online_users)):
        if send_user == online_users[i][1]:
            print('用户已经存在')
            send_user = '' + send_user + '_2'
    if send_user != '':
        if send_user == 'no':
            send_user = addr[0] + ':' + str(addr[1])
        online_users.append((conn, send_user, addr))
        print('建立聊天连接：', addr, ': ', send_user, end='')
        u = get_users()
        print('\n', u)
        self.get_mes(u, addr)
        try:
            while True:
                message = conn.recv(1024)
                message = message.decode('utf-8')
                print(message)
                self.get_mes(message, addr)  # 保存信息到队列
            conn.close()
        except:
            print(send_user + ' Connection lose')
            self.del_user(conn, addr)  # 将断开用户移出users
```

当有客户端给服务端发送消息的时候调用该函数。由于用户刚登陆也会发消息给服务端，所以先对这类消息进行处理，将新的用户添加到 online_user 的列表中。接着用一个 while 循环接收来自该用户的消息，并把消息存放在消息队列中。如果建立连接失败，

意味着该客户端的连接已经断开，说明客户端已经下线，将当前用户从在线用户列表中删除。

- PhotoServer 图片类

```
class PhotoServer(threading.Thread):
    """重写聊天图片线程类，继承threading.Thread"""

    def __init__(self, port):
        threading.Thread.__init__(self)
        self.ADDR = ('127.0.0.1', port)
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.photo = '.\\server_photo_cache\\'
```

```
def run(self):  # 启动线程
    self.s.bind(self.ADDR)
    self.s.listen(5)
    print('正在运行Photo Server的线程')
    while True:
        conn, addr = self.s.accept()
        print('conn=', conn, 'addr=', addr)
        t2 = threading.Thread(target=self.recievePhoto, args=(conn, addr))
        t2.start()
    self.s.close()
```

首先在 init 里面重新初始化相关参数，定义图片类的 IP+PORT 地址，并定义根据地址定义 socket 变量。接着在启动线程的函数中，将 socket 变量与地址关联，同时最多允许接收 5 个。调用 receivePhoto 函数来接收来自客户端的消息，并且只在有图片消息传来的时候才会调用改函数。

```
# 处理图片
def recievePhoto(self, conn, addr):
    while True:
        picture = conn.recv(1024)
        picture = picture.decode('utf-8')
        print('从{0}接收照片：{1}'.format(addr, picture))
        if picture == 'quit':
            break
        action = picture.split()[0]  # 判断是接收还是发送
        if action == 'get':
            self.sendPhoto(picture, conn)
        elif action == 'put':
            self.recvPhoto(picture, conn)
    conn.close()
    print('---')
```

将客户端发来的图片类消息进行解析，如果第一个字符是 get，意味着要发送图片给客户端，因此调用 sendPhoto 函数；如果第一个字符是 put，意味着要接收客户端发来的图片，因此调用 recvPhoto 函数

① sendPhoto 函数

```
# 将缓存中的图片发送给客户端
def sendPhoto(self, picture, conn):
    print(picture)
    pic_name = picture.split()[1]
    pic_path = self.photo + pic_name
    print('开始发送图片')
    file = open(pic_path, 'rb')
    while True:
        pic = file.read(1024)
        if not pic:
            break
        conn.send(pic)
    time.sleep(1)  # 延时
    conn.send('EOF'.encode('utf-8'))
    print('图片发送成功')
```

将服务端接收到的客户端要求下载的图片名称提取出，然后在服务器缓存中找到对应文件，将其以二进制形式发送给客户端，以完成图片发送。

② recvPhoto 函数

```
# 将上传的图片存放在缓冲中
def recvPhoto(self, picture, conn):
    print(picture)
    pic_name = picture.split()[1]
    pic_path = self.photo + pic_name
    print('开始上传图片')
    file = open(pic_path, 'wb+')
    while True:
        pic = conn.recv(1024)
        if pic == 'EOF'.encode('utf-8'):
            print('图片上传成功')
            break
        print(file, ': ', pic)
        file.write(pic)
        file.flush()
    file.close()
```

将服务端接收到的图片名称提取出，作为缓存图片的名称，然后以二进制写入指定的服务器缓存文件夹中。

- FileServer 文件类

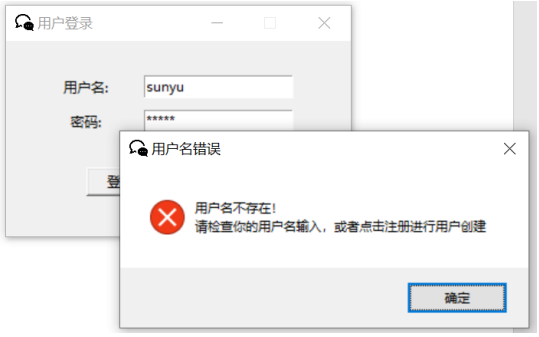
文件服务端的线程类与图片类思路一样，也是通过判断客户端发来的是 put 还是 get 指令，从而进行上传文件或下载文件操作。具体内容在报告里不再赘述，具体代码见源码

五、结果展示

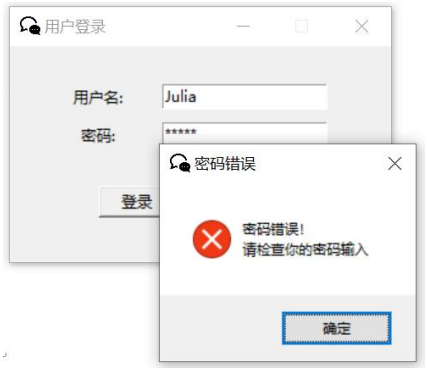
1、登录界面

1）运行开始：

2）用户名不存在：



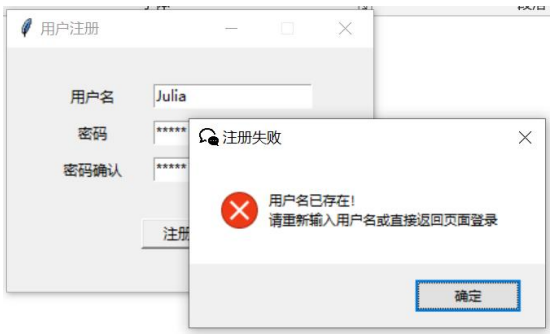
密码错误：



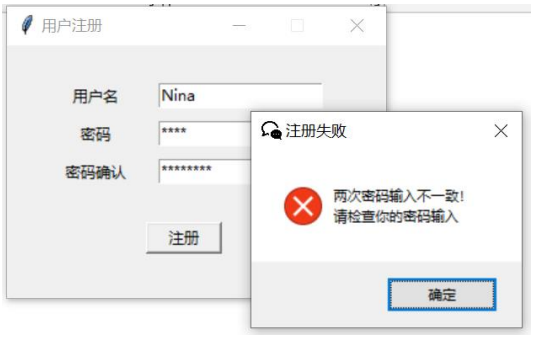
2、注册界面

1）运行开始

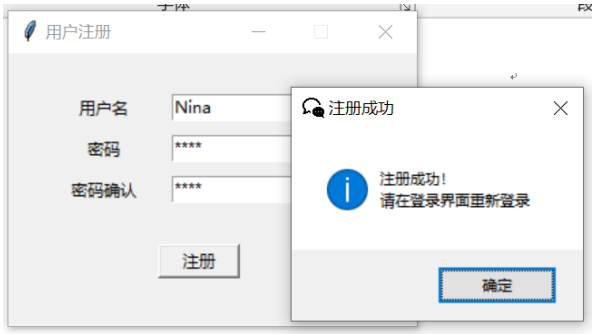
2）用户名已存在：



密码输入不一致：



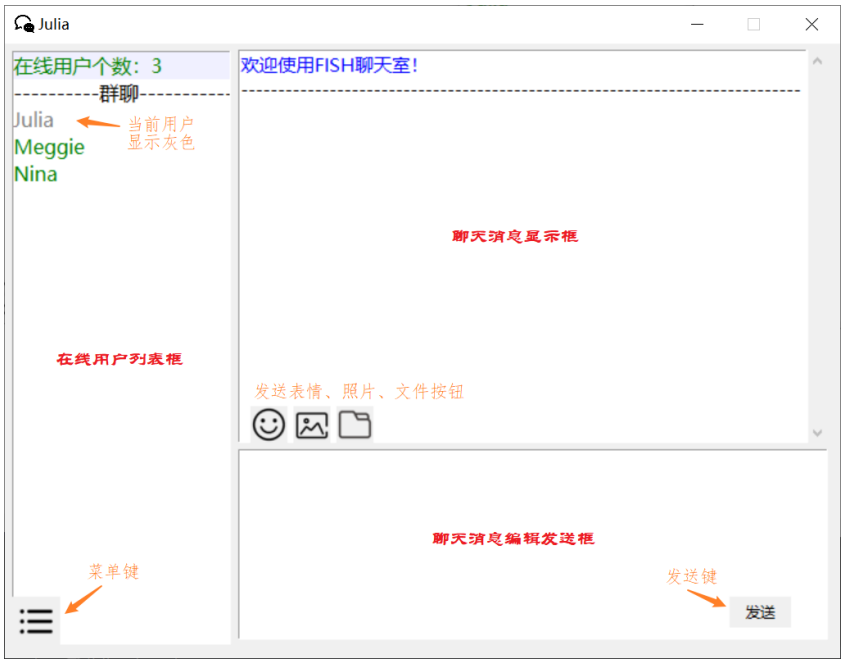
3）注册成功：



3、聊天界面（内含文字、表情、图片）

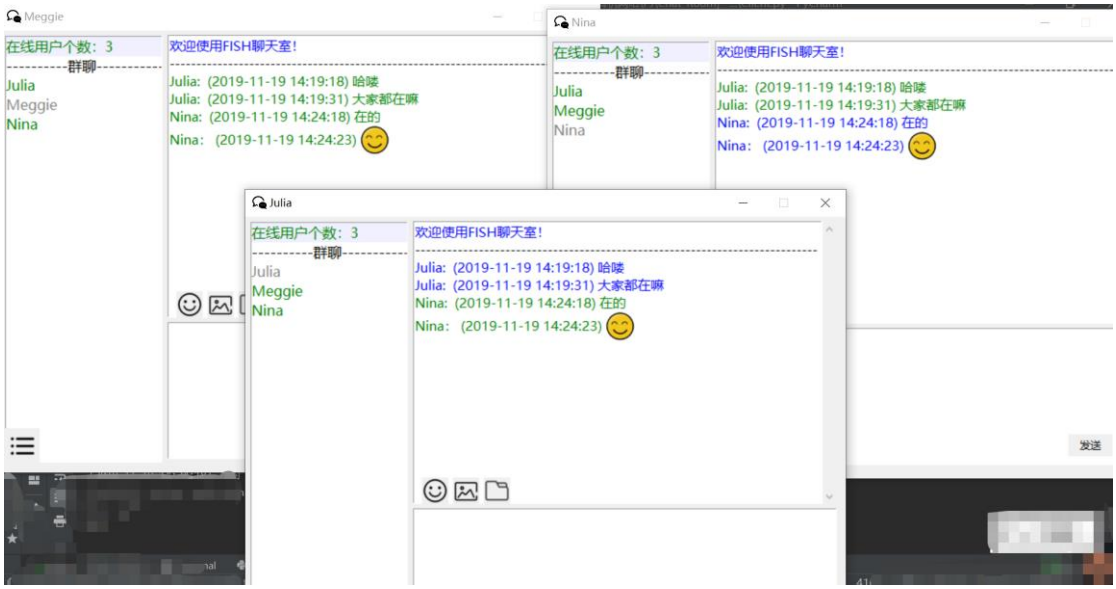
此处以三个用户 Julia, Meggie, Nina 在线为例，展示本聊天器各项功能。

1）开始运行



2）群聊

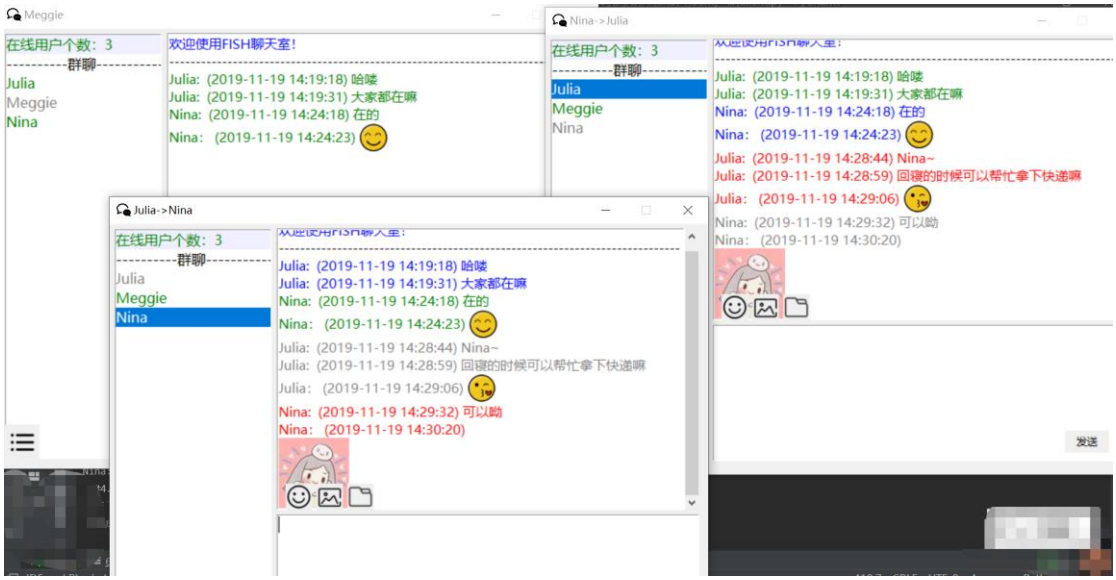
如下图，发送消息的为 Julia 和 Nina。在群聊模式下，对于每一个用户来说，自己发送的消息显示为蓝色，别人发送的消息为绿色。消息会显示发送者和发送时间。



2）私聊

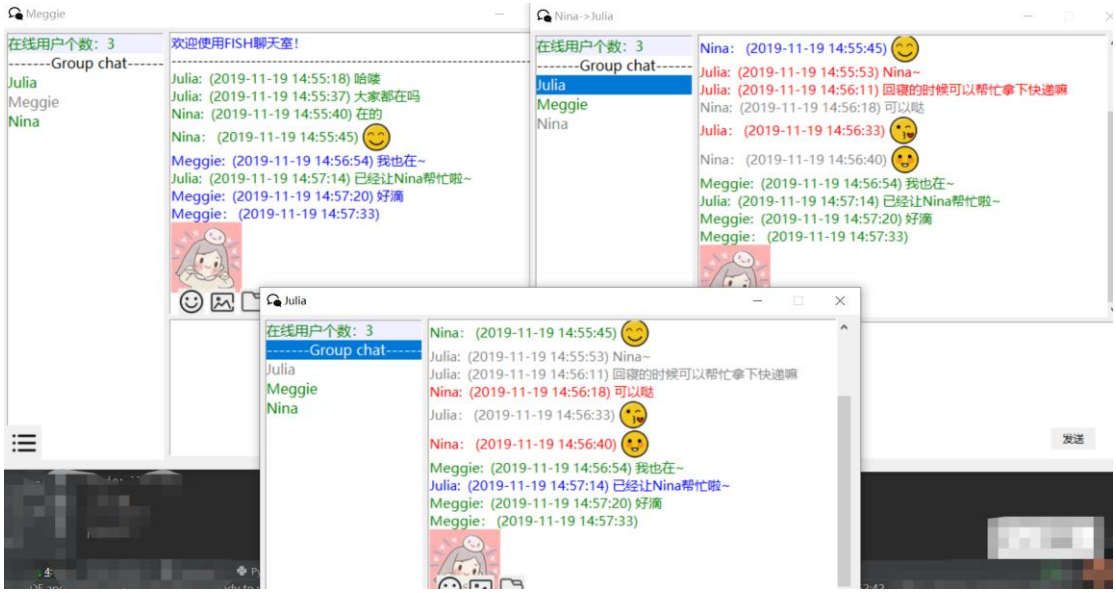
Julia 获知 Nina 在的时候就在用户列表点击 Nina（此时窗口标题变为发送者->接收者），发送私聊消息，在发送者的窗口，发送的私聊消息显示为灰色。而 Nina 收到了红色的消息，表明该消息时来自 Julia 的私聊消息， Nina 可以点击用户列表中的 Julia 并进行回复。

可以看到，此时 Meggie 无法查看另外两人的私聊消息。



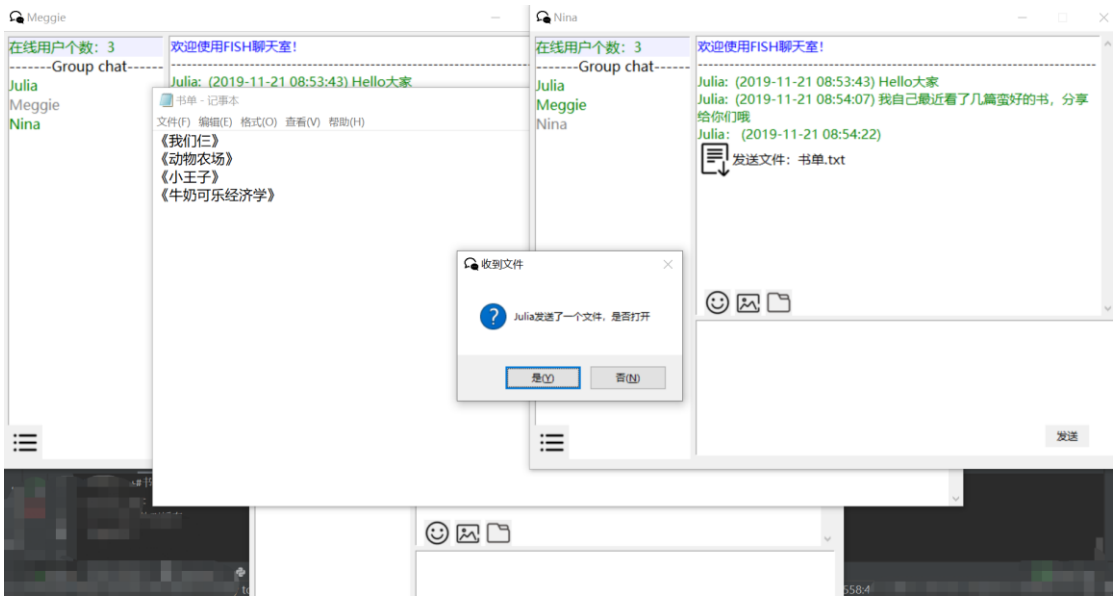
3）重回群聊

此时 Meggie 也在群聊界面中回复在，同样所有人可以收到该消息。Julia 可以点击用户列表中的 Group Chat 以回到群聊。此时 Julia 再回复信息又重新符合 1）中描述的特点。



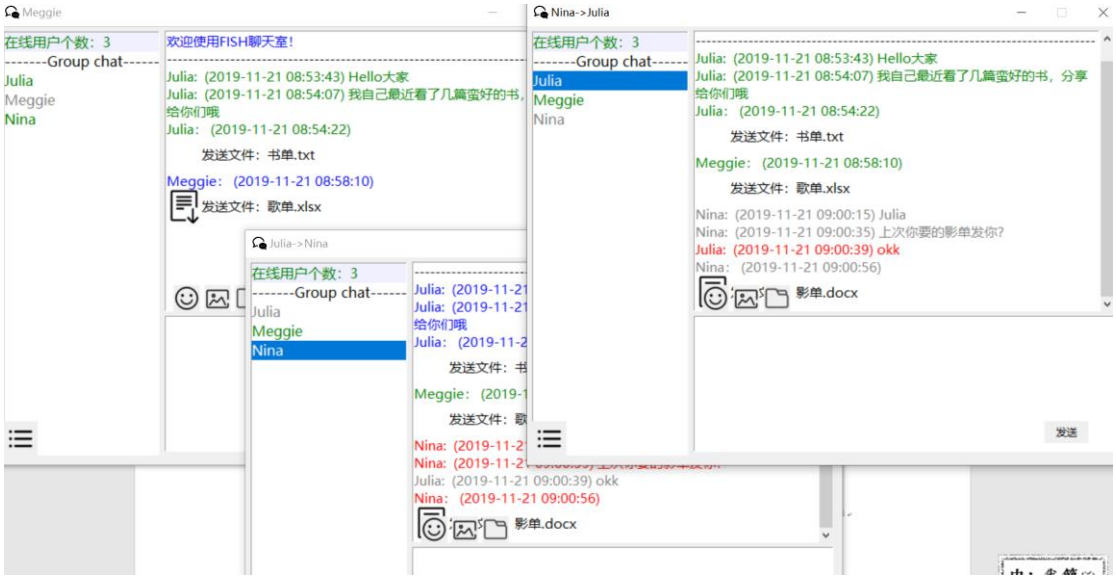
4、传输文件

如下图，当 Julia 群发了一个 txt 文件后，会在消息框中显示，并弹出选择框，选择是否立即查看该文件，如果选是，则会立即打开该文件。



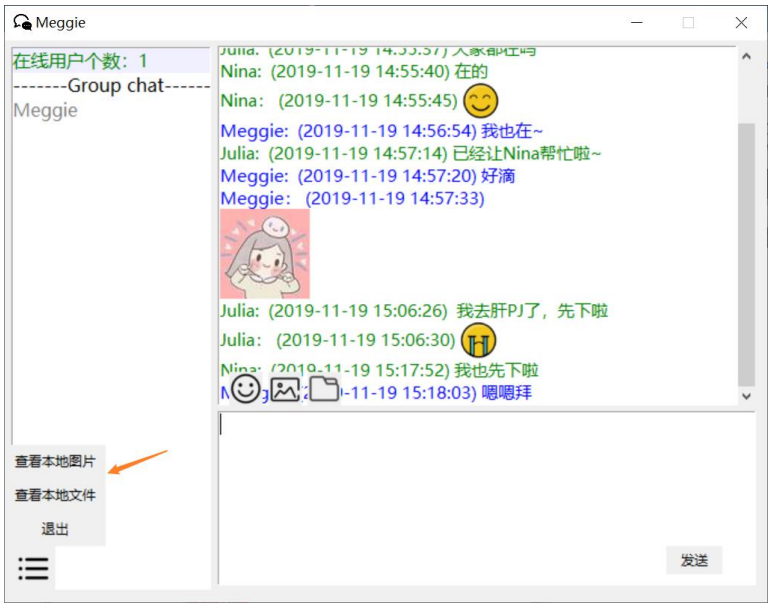
Meggie 分享 xlxs 文件：

文件同样也是私聊对象之外的其他人无法接收：

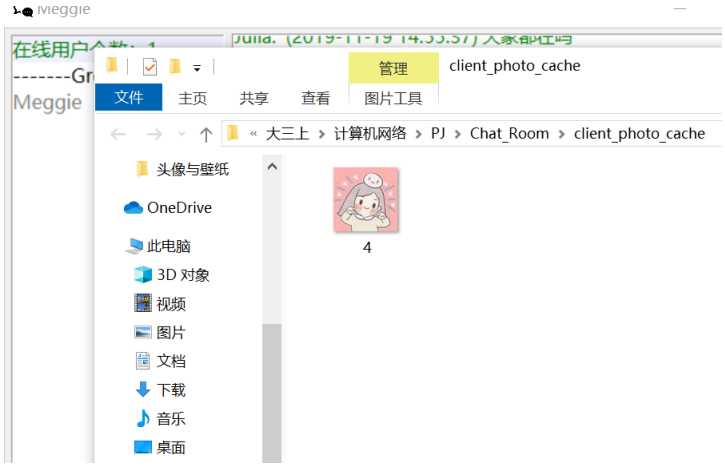


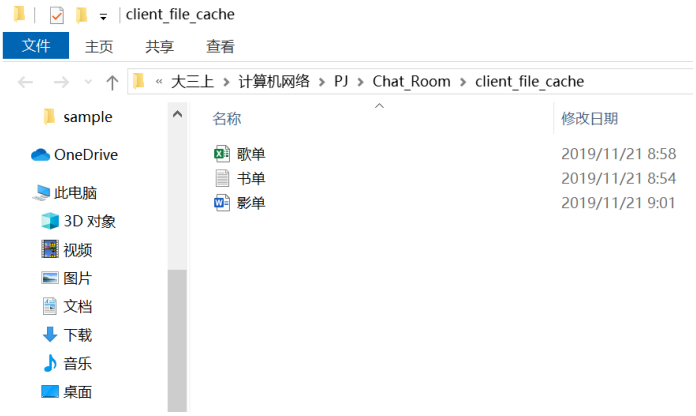
5、查看本地缓存

可以从下面图片箭头处查看聊天缓存的图片和文件



查看本地图片和文件时，会弹出如下窗口，可以看到，文件夹里分别缓存了刚刚发送的图片和文件：



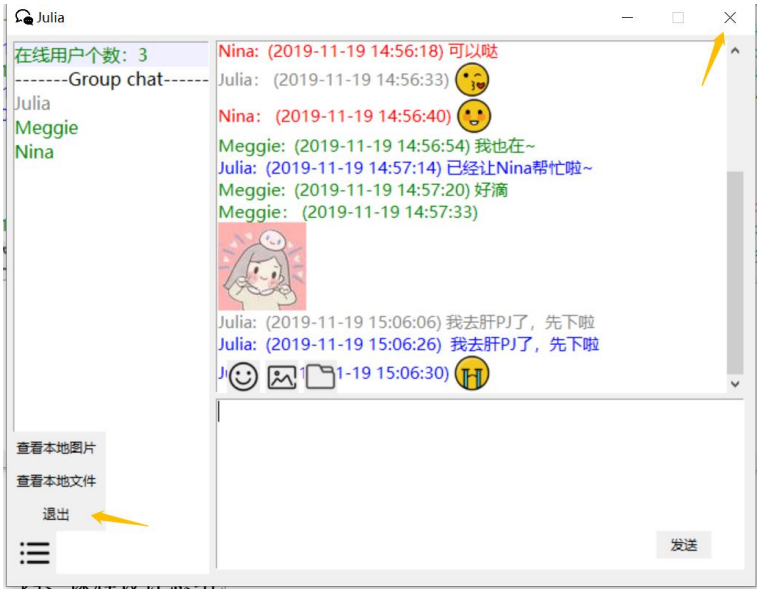


1）查看本地文件

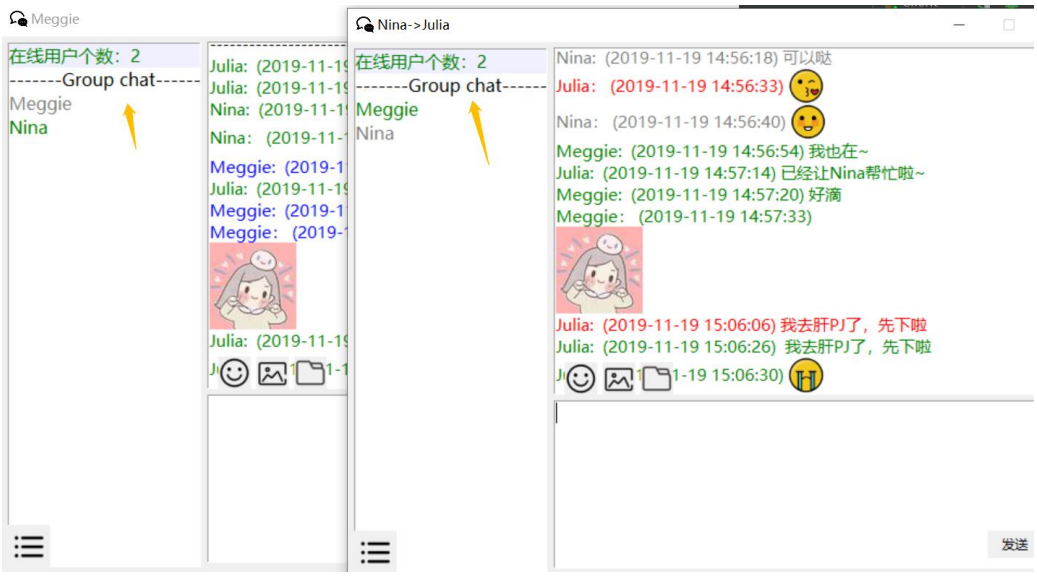
当点击查看文件时，会弹出如下窗口，可以看到，文件夹里缓存了刚刚发送的图片。

6、退出聊天室

如下图所示两个地方均可退出聊天室，即断开连接。



Julia 退出聊天室后，其余两人的在线用户列表发生了变化：



六、课程设计总结

本次课程设计历时四个周终于完成。从选题开始，觉得聊天室是一个比较有意思的方向，于是就尝试完成。我的完成课程设计的大方向是先写界面，后实现客户端与服务端的连接传值；在小方面是按照功能分块，即登录、注册、聊天室（其中细分文本信息、表情信息、图片信息、文件信息）、在线用户列表、简易菜单键。大致界面完成用了一周左右的时间，后面两周专注于实现 socket 连接客户端和服务端并实现多线程的功能。

本聊天器基本上完成了聊天器所需要的基础功能，对日常聊天发送图片文件是足够的，有一定的意义和价值。但是由于对 tkinter 了解程度有限，所以可视化界面做的较为简陋，后期可以进行优化。另外没有对聊天记录进行缓存，导致每次登出后再登入会丢失之前的聊天信息，后期优化可以纳入数据库有关知识进行更多的缓存，以达到更接近真实聊天器的目的。

通过本次实验，我对 socketAPI 实现有了更深层的了解，同时也对应用层的运行机制有了更深的认知。也学到了很多 python 和 tkinter 包的知识。